PENDIX A

```
scnPtr = img + (S.yOfsM+S.yWinM-1)*pixPerScn + S.xOfsM;
if (levels==8) {
  if (S.segmntMode==3) { /* directed acuity mode 3; pack pixels */
    /* S.xWinM and S.yWinM known to be mod 8 & mod 2, respectively */
    for (j=0; j<S.yWinM; j+=2, scnPtr-=2*pixPerScn) {
      pxlPtr = scnPtr;
      for (i=0; i<S.xWinM; i+=2, pxlPtr+=2) {
        bA = (int)*pxlPtr;
        bB = (int)*(pxlPtr+1);
        bC = (int)*(pxlPtr-pixPerScn);
        bD = (int)*(pxlPtr-pixPerScn+1);

        /* bit 0 is set if halftone (&1) */
        /* bit 1 is set if verticle (&2) */
        /* code for diagionals is same value in 224 and 28 locs */
        /* diagionals made only when bit d is blacker than a,b,c */
        vector = 0;
        if (bA&1) vector |= 8; /* halftone A */
        if (bB&1) vector |= 4; /* halftone B */
        if (bC&1) vector |= 2; /* halftone C */
        if (bD&1) vector |= 1; /* halftone D */
        switch (vector) {
          case 0:if/*(((bD&224)<(bB&224))&&((bD&224)<(bC&224)))*/(0) {
                   byt = (bD&227)|((bD>>3)&28); break;
                 } else {
                   byt = (bA&2)? (bA&227)|((bB>>3)&28)
                                 : (bA&227)|((bC>>3)&28);
                   byt = grad(byt); break;
                 }
          case 1:  byt = (bA&2)? (bA&227)|((bB>>3)&28)
                               : (bA&227)|((bC>>3)&28);
                   byt = grad(byt); break;
          case 2:  byt = (bB&2)? (bA&224)|((bB>>3)&28)|(bB&3)
                               : (bA&224)|m28(bA)|(bB&3);
                   byt = grad(byt); break;
          case 3:  byt = (bA&2)? (bA&227)|((bB>>3)&28)
                               : (bA&227)|m28(bA);
                   byt = grad(byt); break;
          case 4:  byt = (bC&2)? (bA&224)|m28(bA)|(bC&3)
                               : (bA&224)|((bC>>3)&28)|(bC&3);
                   byt = grad(byt); .break;
          case 5:  byt = (bA&2)? (bA&227)|m28(bA)
                               : (bA&227)|((bC>>3)&28);
                   byt = grad(byt); break;
          case 6:  byt =         bA&227|m28(bA); break;
          case 7:  byt =         bA&227|m28(bA);
                   byt = grad(byt); break;

          case 8:if/*(((bD&224)<(bB&224))&&((bD&224)<(bC&224)))*/(0)
                   byt = (bD&227)|((bD>>3)&28); break;
                 } else {
                   byt = (bD&2)? m224(bC)|((bB>>3)&28)|(bD&3)
                               : m224(bB)|((bC>>3)&28)|(bD&3);
                   byt = grad(byt); break;
                 }
          case 9:  byt = (bB&2)? m224(bC)|((bB>>3)&28)|(bB&3)
                               : m224(bB)|((bC>>3)&28)|(bB&3);
                   byt = grad(byt); break;
          case 10:
          case 11: byt =         m224(bB)|((bB>>3)&28)|(2);
                   byt = grad(byt); break;
          case 12:
          case 13: byt =         m224(bC)|((bC>>3)&28);
```

```
                    byt = grad(byt); break;
                case 14: byt =          (bD&227)|((l  >3)&28); break;
                case 15: byt =          bA; break;
              }
              temp1 = (uch)byt;
              putc(temp1, stmPtr); /* even is lineart */
            }
          }
        } else { /* mode must be 0,1 or 2 */
          /* S.xWinM known to be mod 4 */
          for (j=0; j<S.yWinM; j++, scnPtr-=pixPerScn) {
            pxlPtr = scnPtr;
            for (i=0; i<S.xWinM; i++, pxlPtr++) {
              temp1 = *pxlPtr;
              putc(temp1, stmPtr); /* even is lineart */
            }
          }
        }
      } else { /* dont do segmentation for quants less than 8 */
        /* quantize the value, then map into a byte, 0 - 255 */
        qdbl0 = (dbl)(1<<levels);
        qdbl1  = (qdbl0 - .000001) / 255.0;
        qdbl2  = 255.0 / (qdbl0-1.0);
        /* S.xWinM known to be mod 4 */
        for (j=0; j<S.yWinM; j++, scnPtr-=pixPerScn) {
          pxlPtr = scnPtr;
          for (i=0; i<S.xWinM; i++, pxlPtr++) {
            temp1 = *pxlPtr;
            if ((temp1==255)||(temp1==0)) {
              putc(temp1, stmPtr);
            } else {
              qdbl3 = (dbl)temp1;
              qdbl4 = (floor(qdbl3*qdbl1))*qdbl2;
              putc(((unsigned char)qdbl4), stmPtr);
            }
          }
        }
      }
      fclose(stmPtr);
    }


it
ad(byte)
it byte;
 return byte; }

it
adl(byte)
it byte;
 /* returns byte with entrys not equal; ie a gradient */
      int b1, b2, b3, b4, b5, vector;

      b1 = byte&224; b2 = (byte&28)<<3;
      if (!(b1==b2)) {
        return byte;
      } else {
        vector = b1>>5;
        switch (vector) {
          case 7: b3=24; break; /* 224 */
          case 6: b3=28; break; /* 192 */
          case 5: b3=24; break; /* 160 */
          case 4: b3=20; break; /* 128 */
```

```
            case 3: b3=8;   break; /* 96 */
            case 2: b3=4;   break; /* 64 */
            case 1: b3=0;   break; /* 32 */
            case 0: b3=4;   break; /* 0 */
          }
          b4 = byte&3;
          b5 = b1|b3|b4;
          return b5;
        }


nt
28(byte)
nt byte;

        if (byte&128) return 28; else return 0;


-
-24(byte)
nt byte;

        if (byte&128) return 224; else return 0;
```

```
if mode==3 { /* disassemble 4x4 bytes into 8,8,8 pixels */
  srcPtr = srcStrt;
  ofsPtr = srcStrt + xwin*2; /* offset by one raster */
  nxtPtr = srcStrt + xwin*4; /* offset by two rasters */
  lstPtr = srcStrt - xwin*4; /* offset by two rasters */
  for (i=0; i<ywin; i++) {
```

PENDIX B

```
    for (j=0; j<xwin; j++) {

      byteP = *srcPtr;
      byteQ = *(srcPtr+2);
      byteR = *nxtPtr;
      byteS = *(nxtPtr+2);
      byteU = *lstPtr; /* U & V have been unpacked last raster */
      byteV = *(srcPtr-2);
      crnrP = *(lstPtr-2);
      crnrQ = *(lstPtr+2);
      crnrR = *(nxtPtr-2);
      crnrS = *(nxtPtr+2);
      dglPS = ((crnrP&1)&&(crnrS&1)); /* both PS corners lineart */
      dglQR = ((crnrQ&1)&&(crnrR&1)); /* both QR corners lineart */
      /* lsb indicates lineart; next lsb indicates horizontal */
      if (byteP&1) { /* p is lineart - gen pqrs as lineart pixels */
        tmpp = (byteP&224);
        pxlp = tmpp|(tmpp>>3);
        if ( ((byteQ&2)^(byteR&2))      /* one vert, the other horz */
            &&(byteQ&1)&&(byteR&1)      /* and both sides are lineart */
            &&(byteU&1)&&(byteV&1)      /* and four sides are lineart */
            &&((dglPS) || (dglQR)) ) { /* and either ps or qr lineart *
          /* if both diags lineart, replicate lowest along diag. */
          /* if only one diag lineart, replicate in dir of diag. */
          pxla = pxlp|1;
          tbot = byteP&28;
          pbot = tbot|(tbot<<3)|1;
          if ((dglPS)&&(dglQR)) { /* both diagionals lineart */
            /* replicate the lowest */
            if (pxla<pbot) { /* replicate a to d; b or c intorpolated *
              pxld = pxla;
              if (byteP&2) { /* p horz; bot of p goes to c; b intrp */
                tmpq = (byteQ&224);
                pxlq = tmpq|(tmpq>>3);
                pxlb = ((pxlp+pxlq)/2)|1;
                pxlc = pbot;
              } else { /* p vert; bot of p goes to b; c intrp */
                tmpr = (byteR&224);
                pxlr = tmpr|(tmpr>>3);
                pxlc = ((pxlp+pxlr)/2)|1;
                pxlb = pbot;
              }
            } else { /* replicate b to c; d intorpolated */
              pxlb = pbot;
              pxlc = pbot;
              tmpq = (byteQ&224);
              pxlq = tmpq|(tmpq>>3);
              tmpr = (byteR&224);
              pxlr = tmpr|(tmpr>>3);
              pxld = ((pxlq+pxlr)/2)|1;
            }
          } else { /* only one diagional PS or QR is lineart */
            if (dglPS) { /* only diagional PS is lineart */
              /* replicate along PS direction */
              pxld = pxla;
              if (byteP&2) { /* p is horz */
                pxlc = pbot;
```

```
            tmpq = byteQ&224;
            pxlq = tmpq|(tmpq>>3);
            pxlb = ((pxlp+pxlq)/2)|1;
          } else { /* p is vert */
            pxlb = pbot;
            tmpr = (byteR&224);
            pxlr = tmpr|(tmpr>>3);
            pxlc = ((pxlp+pxlr)/2)|1;
          }
        } else { /* only diagional QR is lineart */
          /* replicate along QR direction */
          pxlb = pbot;
          pxlc = pbot;
          tmpq = (byteQ&224);
          pxlq = tmpq|(tmpq>>3);
          tmpr = (byteR&224);
          pxlr = tmpr|(tmpr>>3);
          pxld = ((pxlq+pxlr)/2)|1;
        }
      }
      i=i; /* breakpoint */
    } else {
      pxla = pxlp|1;
      if (byteP&2) { /* p is horz */
        tmpc = byteP&28;
        pxlc = tmpc|(tmpc<<3)|1;
        if (byteQ&1) { /* q is lineart */
          tmpq = (byteQ&224);
          pxlq = tmpq|(tmpq>>3);
          if (byteQ&2) { /* q is horz */
            tmqs = byteQ&28;
            pxqs = tmqs|(tmqs<<3);
          } else { /* q is verticle */
            if (byteS&1) { /* s is lineart */
              tmps = (byteS&224);
              pxls = tmps|(tmps>>3);
              pxqs = (pxlq+pxls)/2;
            } else { /* s is halftone */
              pxqs = pxlq;
            }
          }
          pxlb = ((pxlp+pxlq)/2)|1;
          pxld = ((pxlc+pxqs)/2)|1;
        } else { /* q is halftone */
          pxlb = pxla;
          pxld = pxlc;
        }
      } else { /* p is vert */
        tmpb = byteP&28;
        pxlb = tmpb|(tmpb<<3)|1;
        if (byteR&1) { /* r is lineart */
          tmpr = (byteR&224);
          pxlr = tmpr|(tmpr>>3);
          if (byteR&2) { /* r is horz */
            if (byteS&1) { /* s is lineart */
              tmps = (byteS&224);
              pxls = tmps|(tmps>>3);
              pxrs = (pxlr+pxls)/2;
            } else { /* s is halftone */
              pxrs = pxlr;
            }
          } else { /* r is verticle */
            tmrs = byteR&28;
```

```
                pxlc = ((pxlp+pxlr)/2)|1;
                pxld = ((pxlb+pxrs)/2)|1;
              } else { /* r is halftone */
                pxlc = pxla;
                pxld = pxlb;
              }
            }
          }
        } else { /* p is halftone - generate pqrs as halftone pixels */
          pxlp =                    byteP&254;
          pxlq = (byteQ&1)? pxlp : byteQ&254; /* q lineart? */
          pxlr = (byteR&1)? pxlp : byteR&254; /* r lineart? */
          pxls = (byteS&1)? pxlp : byteS&254; /* s lineart? */
          pxla =   pxlp;
          pxlb = ((pxlp+pxlq)/2);
          pxlc = ((pxlp+pxlr)/2);
          pxld = ((pxlp+pxlq+pxlr+pxls)/4);
        }
        *(srcPtr+0) = (uch)pxla;
        *(srcPtr+1) = (uch)pxlb;
        *(ofsPtr+0) = (uch)pxlc;
        *(ofsPtr+1) = (uch)pxld;

        srcPtr += 2; /* advance by one source pixel */
        ofsPtr += 2; /* advance by one source pixel */
        nxtPtr += 2; /* advance by one source pixel */
        lstPtr += 2; /* advance by one source pixel */
      }
      srcPtr += xwin*2; /* advance by one more source raster */
      ofsPtr += xwin*2; /* advance by one more source raster */
      nxtPtr += xwin*2; /* advance by one more source raster */
      lstPtr += xwin*2; /* advance by one more source raster */
    }
  }
}
```